Received May 24, 2020, accepted June 16, 2020, date of publication June 22, 2020, date of current version July 8, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3004258

ECSD: Enhanced Compromised Switch Detection in an SDN-Based Cloud Through Multivariate Time-Series Analysis

PHUC TRINH DINH[1], **(Member, IEEE), AND MINHO PARK**[1,2], **(Member, IEEE)**
[1]Department of Information Communication, Materials, and Chemistry Convergence Technology, Soongsil University, Seoul 156-743, South Korea
[2]School of Electronic Engineering, Soongsil University, Seoul 156-743, South Korea

Corresponding author: Minho Park (mhp@ssu.ac.kr)

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (Ministry of Science and ICT) (No.2018-0-00254, SDN security technology development).

ABSTRACT Nowadays, Software-Defined Networks (SDNs) are increasingly being used in many practical settings, posing a variety of security risks, such as compromised switches. Once a switch is compromised by an attacker, the switch may be either malfunctioning or misconfigured, displaying some abnormal network behaviors, e.g., delaying, dropping, adding, or modifying the traffic. In our previous work, we proposed an efficient scheme for detecting compromised SDN switches based on chaotic analysis of network traffic using an autoregressive-integrated-moving-average model. This scheme showed good results overall; however, it still showed high false-alarm rates due to a hard-set threshold. In this paper, we propose an enhanced scheme to detect compromised SDN switches effectively and reliably. The scheme consists of two phases (online and offline), leveraging the advantages of a stochastic recurrent neural network variant of multivariate time-series-based anomaly detection. Our main idea is to capture the normal patterns of multivariate time series by learning strong representations with the key techniques, such as planar normalizing flow and stochastic variable connection, then reconstruct input data by the representations, and use the reconstruction probabilities to find anomalies. Evaluation results of our proposed scheme yield outstanding performance in comparison with our previous work and other solutions.

INDEX TERMS Network security, software defined networking, distributed cloud computing, SDN compromised switch, anomaly detection, machine learning, network function virtualization.

I. INTRODUCTION

Recently, the trend of integrating Software-Defined Networking (SDN) [1] with Network Functions Virtualization (NFV) [2] (also known as the software-defined NFV architecture) to accomplish various network control and management goals has seen substantial growth. Through NFV, SDN can dynamically create a virtual service environment for a service chain. Consequently, the dedicated hardware and complicated labor required for providing a new service request are avoided. Along with the use of SDN, NFV further enables real-time and dynamic function provisioning, along with flexible traffic forwarding. Moreover, service function chaining (SFC) technology was born to chain multiple services (NIDS, firewall, load balancing, etc.) as a network flow [3]. Figure 1 illustrates a typical SDN-based cloud environment prototype including three technologies, SDN, NFV, and cloud platform.

The associate editor coordinating the review of this manuscript and approving it for publication was Jing Bi.

A. PROBLEM STATEMENTS

Switches in the data plane have no intelligence; they simply send raw data packets to the controller. Unfortunately, this behavior introduces a serious vulnerability that can be exploited by an attacker. A malicious or faulty SDN switch could severely disrupt network communications. In particular, colluding malicious switches that try to conceal their misbehavior are more challenging to detect. The anomalous behaviors at an SDN switch can be divided into five categories:

- Traffic Loss: a compromised switch can drop traffic randomly or selectively.
- Traffic Misroute: the traffic of a compromised switch can be forwarded to a wrong address.

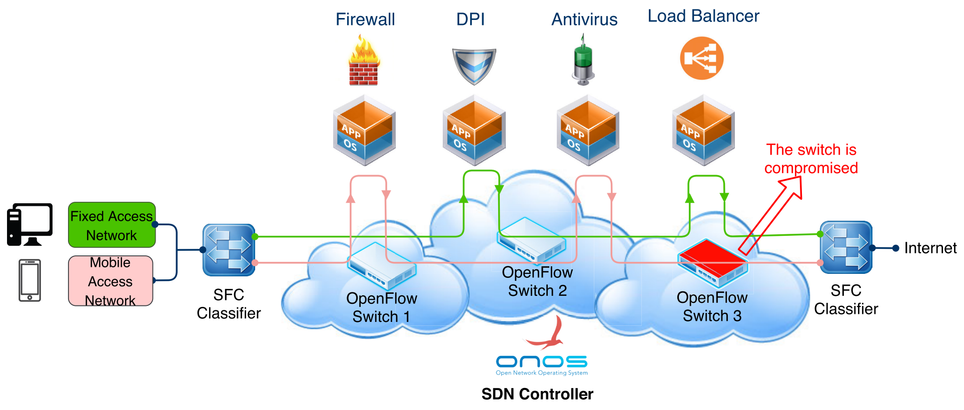

FIGURE 1. **SDN-based cloud environment prototype including a compromised switch.**

- Traffic Modification: The traffic contents might be modified by a compromised switch.
- Traffic Reorder: a compromised switch may change the order of packets while still being valid in terms of content, delay, and routes. Continuous reordering of TCP packets can significantly deteriorate the TCP throughput [4].
- Traffic Delay: a compromised switch may delay traffic in order to increase jitter, which raises full of problems with time-sensitive traffic. Moreover, delaying traffic for a TCP stream triggers spurious timeouts and unjustified retransmissions, resulting in grave damage to the TCP throughput [13].

From the compromised switch behaviors mentioned above, an attacker can compromise a switch by dropping, slowing down, or misrouting network traffic [13], or by executing flooding attacks against the control plane, either by replaying or spoofing packet_in messages [5]. The damaging effects of compromised switches can cause great difficulties in any network system. In addition, to the best of our knowledge, and as seen in [5], [13]–[15], not many studies have efficiently solved these critical problems.

B. OUR PROPOSAL

To resolve the serious issues given above, in this paper, we present a concrete proposal with a novel mechanism that monitors, checks and detects anomalous behaviors of SDN switch traffic. The proposed mechanism applies a technique that is a multivariate time series anomaly detection through a stochastic recurrent neural network variant, and then uses a dynamic threshold selection to automatically set the optimal threshold to differentiate between normal and abnormal SDN switch traffic.

C. CONTRIBUTION

Our major contributions can be listed as follows:

- First, we propose a new approach, which is a multivariate time-series technique to keep track of and detect anomalous behaviors of SDN switch traffic quickly.
- Second, we propose a complete scheme, Enhanced Compromised Switch Detection (ECSD), which applied the multivariate time-series technique to detect compromised switch attacks effectively.
- Next, we investigate the vulnerabilities of the SDN switches on the cloud and the present common types of attacks in a cloud-based SDN environment in a distributed manner.
- Our experiments were conducted in a real cloud computing environment using OpenStack [42] integrated with the SDN environment using the Open Network Operating System (ONOS) controller [17], but they could be smoothly applied to other controllers.
- Finally, we conducted experimental studies and compared our detection scheme with other existing proposals and some machine-learning-based algorithms that apply the multivariate time-series technique. An extensive comparison of ECSD demonstrates the improved efficiency of our proposed scheme.

The rest of this article is constructed as follows. Section II shows some research related to our work. Section III presents background knowledge about the SDN-based cloud, compromised switches on the cloud, and basics of GRU, VAE, SGVB, and Planar NF. Section IV gives our research rationale, system analyses, and practical design of the proposed security mechanism. Section V focuses on our experimental setup, and the results are analyzed using several evaluation metrics in Section VI. Lastly, Section VII concludes our work and lays out some potential future developments.

II. RELATED WORK

In general, the improvement of SDN security applications and controllers, and the online verification of network restrictions, separately, have been the prime focus of SDN security literature [6], [13]. Intrusion detection systems (IDSs) are important as prime defense techniques to protect network systems. Thus, Teng *et al.* [9] proposed an intelligent model based on two famous machine-learning algorithms (decision trees (DT) and support vector machine (SVM)), and they did a test on their model on a KDD CUP 1999 dataset. The results demonstrated an accuracy reaching 89%. However, SVMs normally show high computation cost and poor performance. Another IDS work is known as improved SD-WSN framework [10] based on a SDN scheme. This framework addresses the network management, node failure issues and provides a means for flexible data forwarding. However, not many studies of these issues provide effective protection against compromised forwarding devices in the data plane [5], [8]. Attacks from the data plane have posed serious threats to SDN [7]. Using multiple hosts under OpenFlow switches, attackers can disrupt the control plane or learn its behaviors without knowing much information about controller applications. These attacks include DoS, topology poisoning, and side-channel attacks [15]. Some faulty behaviors that originate at SDN switches are traffic loss, traffic fabrication, traffic misrouting, traffic modification, traffic delay, and traffic reordering [13]. Among existing solutions, a proposal called SPHINX [25] has practical implementations [5]. It detects and mitigates security attacks initiated by malicious switches by abstracting the network operations with incremental flow graphs. It also detects attacks as per the policies defined by the administrator and responds accordingly. However, SPHINX still has its own limitations. First, SPHINX cannot detect when a malicious forwarding device is delaying packets [26]. Second, it triggers significant communication overheads, because it gathers statistics of all flows from all switches [27]. A fairly effective scheme called WedgeTail [26], which is the closest work to SPHINX, is known as an effective intrusion prevention system for the data plane of SDNs. This work does not depend on rules pre-defined by an administrator, and it is capable of detecting and responding to all embedded malicious forwarding devices within a reasonable time-frame. However, to make WedgeTail work, the authors of WedgeTail made some assumptions, such as that the control plane itself and the defined policies are trustworthy, but in practice, the control plane is highly vulnerable, which attackers can manipulate to deploy a compromised switch attack. Another work is known as FlowMon [24], in this work, two anomaly detection algorithms are proposed, Packet Droppers and Packet Swappers. To detect malicious switches, the controller analyzes the collected port statistics and the actual forwarding paths. However, FlowMon might be dysfunctional if the dishonest switches provide false information in their statistical reports. Lastly, an online detection mechanism is proposed to find suspicious SDN switches and generate security alerts using security information and event management (SIEM) technology [14]. The technology can provide real-time analysis of security alerts for network managers. However, SIEM only covers some abnormal switches behaviors such as incorrect forwarding, packet manipulating and weight adjusting.

Recently, we proposed a new approach [16] to detect compromised switches using an autoregressive integrated moving average (ARIMA) learning model to predict the numbers of flows and calculating the maximum Lyapunov exponent to analyze the chaotic behaviors of prediction error time-series. If the Lyapunov exponent remains positive for $K = 2$ consecutive times, and some of the parameters which we observed during the attack exceed their thresholds, then the detection scheme raises an alarm. Although this is quite an effective method, it has a high false-alarm rate because of setting hard thresholds, and it has high CPU usage during the attack times due to centralized training only.

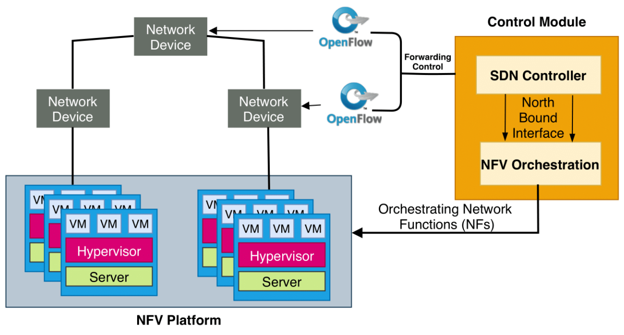

FIGURE 2. SDN-based cloud prototype.

Understanding these difficulties, in this paper, we propose a mechanism to detect and defend SDN compromised cloud-based switches effectively using dynamic thresholds, which brings about high detection rates, low false-alarm rates, and efficiency in resource consumption. Our detection mechanism does not deteriorate network performance. Moreover, switches do not necessarily isolate or turn off for detection. Instead, the administrator can easily launch the detect process when the network is running. Therefore, our detection mechanism is very practical.

III. BACKGROUND KNOWLEDGE

A. SDN-BASED CLOUD

SDN-enabled cloud computing has been emerging as a future SDN-based cloud environment. Many studies have proposed a various methods to not only increase revenue for data center providers but also reduce the round-trip time of tasks for applications, for example, [11], [12]. The integration of NFV and SDN technologies is known as the SDN/NFV architecture [2], as illustrated in Figure 2. It consists of NFV orchestration, a controller platform, forwarding devices, and servers. The SDN controller is responsible for controlling the traffic path using primarily the OpenFlow protocol to communicate with forwarding devices (OpenFlow switch) to impose policies from the control plane to the data plane. Meanwhile, the NFV uses standard computing virtualization technology to consolidate in commodity hardware (i.e., servers or cloud platform (e.g. OpenStack)) to deliver network functions of high bandwidth and high performance with low cost. Hypervisors, which run on servers, primarily focus on supporting VMs that enable them to operate network functions, such as firewalls, proxies, and IDS. The logical control module is composed of the SDN controller [18] and the NFV orchestration system. The NFV orchestration system is responsible for provisioning virtualized network functions and is controlled through standard interfaces or APIs by the SDN controller. After determining the policy requirements and creating network topology, the controller generates optimal function assignments and assigns the functions to certain VMs in the optimized path by translating the logic policy specifications, which is known as a service chain. The NFV orchestration system conducts a service function chain, and the controller instructs the traffic through the required and appropriate sequence of VMs by installing flow rules into forwarding devices.

B. COMPROMISED SWITCHES ON THE CLOUD

Compromised switches are a novel and thorny problem for SDNs. In the two studies [13] and [14], authors formally defined various types of compromised switches attacks, e.g., packet dropping, packet duplicating, packet manipulating, incorrect forwarding, traffic delaying, and traffic reordering, etc. These attacks are stealthy, and they can occur while the compromised switch performs packet forwarding. By controlling the compromised switches to execute one of these attacks, attackers can bring severe problems to the entire network. Moreover, these kinds of attacks can severely deteriorate the TCP throughput [4].

Figure 1 is a typical example to illustrate the harmfulness of a compromised switch. The compromised *OpenFlow switch 3* can drop packets or falsely forward them to its switch neighbors multiple times. It also can delay packets for a while. All these kinds of actions may ruin the operation of the entire network.

C. BASICS OF A VARIANT OF RNN - GRU, VAE, SGVB, AND PLANAR NF

Recurrent neural network (RNN) [22], a well-known method in deep learning, feeds the output from the previous step as the input to the current step, which enables RNN to represent the time dependence. However, RNN fails to learn the long-term dependence for multivariate time-series data due to the complex temporal dependence and stochasticity.

Fortunately, gated recurrent unit (GRU), a variant form of RNN, can learn long-term dependence even without using a memory unit to control the flow of information, because it just exposes the full hidden content without any control. Moreover, GRU does not require a large dataset because of its fewer parameters and simpler structure [19]; therefore, GRU is applied in this study. Variational autoencoder (VAE) is another deep-learning technique for learning latent representations [20], and it has been successfully applied to anomaly detection for seasonal univariate time series [35]. VAE represents a high-dimensional input x_t to a latent representation z_t with a reduced dimension, and then reconstructs x_t by z_t. With a prior $p_\theta(z_t)$ for z_t, x_t is sampled from posterior distributions $p_\theta(x_t|z_t)$. VAE approximates $p_\theta(x_t|z_t)$ using an inference network $q_\phi(z_t|x_t)$, where ϕ and θ are parameters of the inference network (i.e., *qnet*) and the generative network

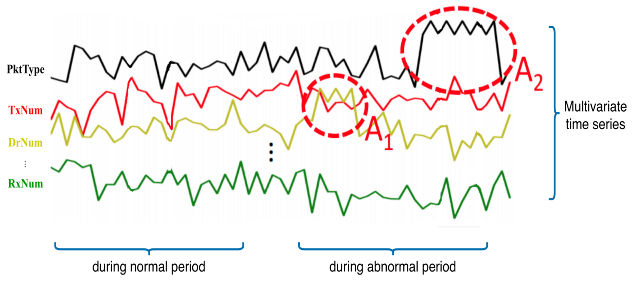

FIGURE 3. SDN switch statistics visualized as a multivariate time-series snippet, with two anomalies A_1 and A_2, marked by red dashed circles. The root causes are the yellow and black time series, respectively.

(i.e., *pnet*), respectively, because it is intractable to compute $p_\theta(x_t|z_t)$.

Stochastic gradient variational bayes (SGVB) [20] is a variational inference algorithm commonly applied in VAE to tune the parameters ϕ and θ and maximizes the evidence of lower bound (ELBO) [28], $\mathcal{L}(x_t)$:

$$\begin{aligned}&\mathcal{L}(x_t)\\&= \mathbb{E}_{q_\phi(z_t|x_t)}[log(p_\theta(x_t|z_t))] - D_{KL}[q_\phi(z_t|x_t)||p_\phi(z_t)]\\&= \mathbb{E}_{q_\phi(z_t|x_t)}[log(p_\theta(x_t|z_t)) + log(p_\theta(z_t)) - log(q_\phi(z_t|x_t))]\end{aligned} \tag{1}$$

Planar normalizing flows (Planar NF) [21] is a transformation technique that transforms $q_\phi(z_t|x_t)$ using the invertible mappings to learn the non-Gaussian posterior density $q_\phi(z_t|x_t)$.

The steps of this combination are shown as follows and in Figure 6. First, GRU is used to capture complex temporal dependencies between multivariate observations in $x-$space. Second, VAE is applied to map observations (i.e., input observations in $x-$space) to stochastic (i.e., $z-$space) variables. Third, to model temporal dependence among stochastic variables explicitly in the latent space, a connection technique called linear Gaussian state-space model (SSM) is applied to connect between stochastic variables and the concatenation of stochastic variables and GRU hidden variables [28]. Lastly, to help stochastic variables in *qnet* (described hereafter) capture complex distributions of input data, Planar NF is also applied to grasp non-Gaussian posterior distributions in latent stochastic space.

The overall model architecture is presented in Figure 6, which is composed of a *qnet* and *pnet*. In the *pnet*, it uses a latent representation $z_{t-T:t}$ (a tuple of probability distributions) to reconstruct the input $x_{t-T:t}$. An accurate representation can reduce the reconstruction loss. The *qnet* is optimized to approximate the *pnet* and obtains quality latent representations [28].

IV. ECSD: ENHANCED COMPROMISED SWITCH DETECTION

In this section, we first describe a rationale for using a multivariate time-series detection technique for our scheme and system design analysis. Second, a workflow system is described, and then the system process logic is presented.

TABLE 1. Notation definitions.

Symbol	type	meaning
z	v	latent representation
$p_\theta(z_t)$	v	prior distributions for z_t
$p_\theta(x_t\|z_t)$	v	x_t is sampled from posterior distribution
$q_\phi(z_t\|x_t)$	v	an inference network
ϕ and θ	p	parameters of the inference network
$\mathcal{L}(x_t)$	v	evidence of lower bound (ELBO)
F	v	a variance ratio for the overall test
$\overline{MST}$	v	mean square error (between groups)
$\overline{MSE}$	v	mean square error (within groups)
Y_{ij}	v	an observation
T_i	v	a group total
G	v	a grand total of all observations
n_i	v	a number in group i
n	v	a total number of observations
$X_{normalized}$	v	a normalized array
x_i	v	a value of x in X
x_{min}	v	the smallest value in X
x_{max}	v	the largest value in X
u*-s, **w***-s, and **b***-s	p	parameters of the corresponding layers
$\tilde{\mathcal{L}}$	v	loss function
f^k	v	invertible mapping functions
S_t	v	anomaly score of x_t
$\overline{F}$	v	a GDP function
γ and β	p	shape and scale parameters of GPD
th	v	an initial threshold of anomaly scores
th_F	v	a final threshold
q	v	a desired probability to observe $S < th$
$\hat{\gamma}$ and $\hat{\beta}$	p	parameters to compute th_F using MLE
N'_{th}	v	the number of S_i
low quantile	c	$< 7\%$
q	c	10^{-4}

Note: v: variable, p: parameter, c:constant.

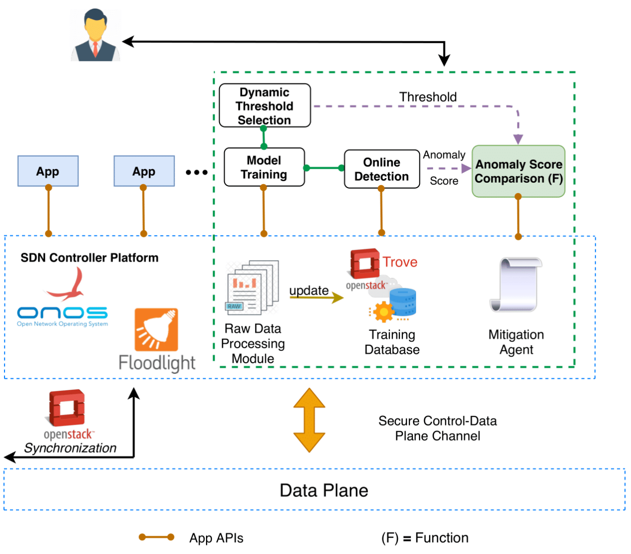

FIGURE 4. Conceptual Architecture.

Lastly, the internal modules of the proposed scheme will be provided in detail. Table 1 summarizes parameters, variables, and constants used in this paper.

A. RATIONALE OF USING A TIME-SERIES TECHNIQUE AND SYSTEM DESIGN ANALYSIS

In the SDN-based cloud, SDN switches are typically monitored with multivariate time series, whose anomaly detection is critical for service quality management. For example, as shown in Figure 3, networking metrics, such as type of packet (PktType), the total number of received packets (RxNum), the total number of transmitted packets (TxNum) and the total number of dropped packets (DrNum), are collected and supervised by a network administrator. When a compromised switch attack is launched, one of the selected features will suddenly change. Thus, we need an intelligent model to discover any changes in the values of multiple time series simultaneously to detect the attack. Therefore, in this paper, a technique called stochastic recurrent neural network [28] is adopted for multivariate time series anomaly detection. The key idea is to capture the normal patterns of multivariate time series by learning strong representations and to use the reconstruction probabilities to determine anomalies using dynamic threshold selection.

A time series involves successive observations which are normally collected at equally spaced timestamps [29]. In this work, we concentrate on multivariate time series, defined as $x = \{x_1, x_2, \ldots, x_N\}$, where N is the length of x, and an observation $x_t \in R^M$ is an M-dimensional vector, at time $t(t \leq N): x_t = [x_1^{(t)}, x_2^{(t)}, \ldots, x_m^{(t)}]$, and $x \in R^{M\times N}$. The $x_{t-T:t}$ $(\in R^{M\times(T+1)})$ is used to denote a sequence of observations $\{x_{t-T}, x_{t-T+1}, \ldots, x_t\}$ from time $t-T$ to t. For multivariate time-series anomaly detection, the objective is to determine whether an observation x_t is anomalous or not. For time-series modeling, historical values are extremely useful for interpreting current data. Thus, a sequence of observations, $x_{t-T:y}$, instead of just x_t, is applied to compute the anomaly. First, the anomaly detection approach returns an anomaly score for x_t, and then the anomaly result can be obtained by comparing with a threshold.

Figure 4 depicts an overview of our proposed framework aimed at detecting a compromised switch in SDN architecture synchronized with an OpenStack [42] controller, and the proposed framework is an extension in the Control Plane. This framework consists of extension components that can be implemented and distributed in both the cloud controller platform (OpenStack) and SDN application plane. We suggest that these components should be set at a dedicated security server in actual deployment in order to deplete the SDN controller processing load. However, in this work we design modules including model training, online detection, and dynamic threshold selection, and locate these modules in the SDN application layer for the convenience, only some of the components are located in the OpenStack controller for synchronization and further actions.

B. WORKFLOW SYSTEM

In this work, our mechanism is situated in the SDN controller and consists of five separate main modules: raw data processing (flow collector, feature extractor, and feature transformation and normalization), model training, online detection, dynamic threshold selection, and mitigation agent, as shown in Figure 5.

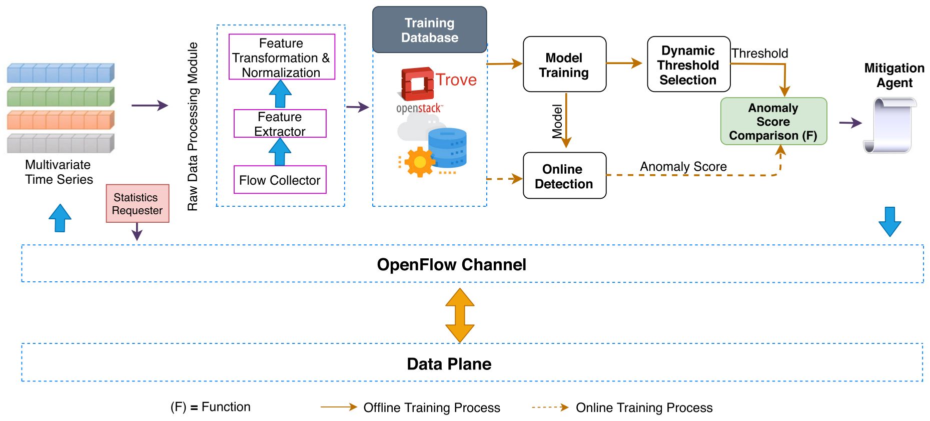

FIGURE 5. Detailed architecture modules and synchronized OpenStack controller located in the SDN application layer.

Raw data preprocessing is a module shared by both model training and online detection. During data preprocessing, the traffic collector reliably collects data on the use of the physical and virtual resources comprising deployed clouds and acquires flow information from OpenFlow switches at a predefined time interval. Next, this data will then be sent to the feature extractor to extract data's attributes. Subsequently, the data is transformed by data standardization, and then it is segmented into sequences through sliding windows [30] of length $T + 1$. After preprocessing, a training multivariate time series, usually spanning a certain period (e.g., a couple of minutes), is sent to the model training module. It learns a model that captures the normal patterns of multivariate time series, and outputs an anomaly score for each observation. These anomaly scores are used by the dynamic threshold selection module to choose an anomaly threshold automatically following the POT method [31]. This model training procedure can be conducted periodically.

The online detection module stores the trained model. A new observation (e.g., x_t at time t) after preprocessing can be fed into the online detection module to get its anomaly score. If the anomaly score of x_t is below the anomaly threshold, x_t will be declared as anomalous, otherwise, it is normal. If the traffic is classified as normal traffic, then it will be normally forwarded to the destination; if not, the traffic will be transferred to the mitigation agent, which will perform dropping or deleting actions if it detects abnormal traffic.

Note that, in our scheme, the training database is continually updated by the attributes of data collected from the above loop. In a preset time, which is being defined by a network administrator, both the model training and online detection will be trained using the updated database. By doing so, the proposed mechanism can adapt well to the various network systems.

C. SYSTEM PROCESS LOGIC

First, a set of observations are assigned to N, in which each observation having M dimensions includes a set of attributes. Moreover, we used $x^{\iota}_{t-T:t}$ ($\in R^{M\times(T+1)}$) to denote a sequence of observations $\{x^{\iota}_{t-T}, x^{\iota}_{t-T+1}, \ldots, x^{\iota}_{t}\}$ from time $t - T$ to t. *preLength* is assigned as the length of current $x^{\iota}_{t-T:t}$, so that it can be used to compare with the length of $x^{\iota}_{t-T:t}$ at the next current time t. If a new observation arrives, then the loop executes its functions consecutively. Before putting the sequence $x^{\iota}_{t-T:t}$ with T consecutive observations as an input for both the offline model training ($\overline{OMT}$) and online detection ($\overline{OD}$) models (more details in section IV.D), we feed $x^{\iota}_{t-T:t}$ into a transformation and normalization ($\overline{TN}$) module to transform, and normalize $x^{\iota}_{t-T:t}$. After being processed in the $\overline{TN}$ module, we have a new sequence $x_{t-T:t}$. Next, after obtaining both *threshold* and *anomaly_score* using different modules (such as $\overline{OMT}$, dynamic threshold selection ($\overline{DNT}$), and $\overline{OD}$), the final decision is made by the following rules:

- If $S_{2t} < threshold$, then the new traffic is labeled as an attack source, therefore, $Flag_t$ is assigned to -1, then, it will be forwarded to the mitigation agent to handle later.
- If $S_{2t} >= threshold$, then the new traffic is labeled as a normal source, $Flag_t$ is assigned to 1, and then, the algorithm continues its loop.

The following Algorithm 1 explains the processes in detail.

D. INTERNAL MODULES

Let us have a closer look into the internal modules of our proposed scheme.

1) TRAFFIC COLLECTOR

First, to get data from the data plane, we run a statistics requester that sends periodic request statistics [32] to an OpenFlow switch and waits for the response statistics (Figure 5). After obtaining the response statistics, the OpenFlow channel sends them to the raw data processing module. Next, a flow collector module simply runs on the SDN controller and collects data from both the OpenFlow switch statistics and the *StatsResponse* messages [32] for a preset period.

2) FEATURE EXTRACTOR

Based on our empirical research and a technique in feature selection called ANOVA (analysis of variance) [33], this module extracts data information from the traffic collector

Algorithm 1 ECSD: Proposed Compromised Switch Defense Scheme

```
$N \longleftarrow$ A set of observations
$x^t = \{x_1^t, x_2^t, \ldots, x_N^t\} \longleftarrow$ A multivariate time-series tuple derived from attributes of an observation
$M \longleftarrow$ each observation has a dimension of $M$
$x_t^t = [x_t^{t1}, x_t^{t2}, \ldots, x_t^{tM}] \longrightarrow x_t^t \in R^M$
$\{x_{t-T}^t, x_{t-T+1}^t, \ldots, x_t^t\} \longrightarrow x_{t-T:t}^t$ $(\in R^{M\times(T+1)})$, from time $t-T$ to $t$
$\overline{TN} \longleftarrow$ Transformation and Normalization
$\overline{OMT} \longleftarrow$ Offline Model Training
$\overline{OD} \longleftarrow$ Online Detection
$\overline{DNT} \longleftarrow$ Dynamic Threshold Selection
$S_{1i}$ $(anomaly\_score) \longleftarrow$ Output of $\overline{OMT}$
$S_{2t}$ $(anomaly\_score) \longleftarrow$ Output of $\overline{OD}$ at time $t$
$preLength = (x_{t-T:t}^t).length()$
loop
    while true:
        if $(x_{t-T:t}^t).length() > preLength$ do
            $preLength+ = 1$:
            Feed $x_{t-T:t}^t$ into $\overline{TN} \longrightarrow x_{t-T:t}$
            Feed $x_{t-T:t}$ into $\overline{OMT} \longrightarrow S_{1i}$
            Feed $S_{1i}$ into $\overline{DNT} \longrightarrow threshold$
            Feed $x_{t-T:t}$ into $\overline{OD} \longrightarrow S_{2t}$
            if $S_{2t} < threshold$ then
                $Flag_t = -1$ (Attack source)
                Forward $x \longrightarrow MitigationAgent$:
            else
                $Flag_t = 1$ (Normal source)
                continue
            end if
        end if
    end while
end loop
```

to take out several attributes, as shown in Table 2. These key features are selected from k best ANOVA F-values using SelectKBest [33].

$$F = \frac{\overline{MST}}{\overline{MSE}}$$

$$\overline{MST} = \frac{\Sigma_{i=1}^{k}(T_i^2)/n_i) - G^2/n}{k-1}$$

$$\overline{MSE} = \frac{\Sigma_{i=1}^{k}\Sigma_{j=1}^{n_i}Y_{ij}^2 - \Sigma_{i=1}^{k}(T_i^2/n_i))}{k}$$

where F is the variance ratio for the overall test, $\overline{MST}$ is the mean square (between groups), $\overline{MSE}$ is the mean square (within groups, residual mean square), Y_{ij} is an observation, T_i is a group total, G is the grand total of all observations, n_i is the number in group i, and n is the total number of observations.

3) FEATURE TRANSFORMATION & NORMALIZATION

At this stage, all the categorical features, which were extracted in the previous stage, are transformed from categorical data into numeric data using the one-hot encoding technique [34]. After the transformation, the data needs to be normalized, we use *max-min normalization* and its formula can be expressed as:

$$X_{normalized} = \frac{x_i - x_{min}}{x_{max} - x_{min}}, \forall x_i \in X \tag{2}$$

where x_{min} is the smallest value in X, and x_{max} is the largest value in X. This module not only significantly improves the model training's predictive power, but also offers the model for faster to run and more easily understood.

TABLE 2. Key Features.

Feature	Description
protocol_type	network protocol
no. of bytes	number of bytes
no_flows	number of flows
no_pkt_flow	number of packets per flow
no_bytes_flow	number of bytes per flow
flow_duration	flow duration
gocp	growth of client ports
PktType	type of packet
network_bandwidth	network bandwidth
PktLeng	payload size in bytes
ProcTime	total time for processing packet
TxNum	total number of transmitted packets
DrNum	total number of dropped packets
RxNum	total number of received packets

4) MODEL TRAINING

Figure 6 illustrates the model architecture of the multivariate time-series learning model proposed in [28], which is adopted in this paper. Both the *qnet* and *pnet* start training at the same time by tuning the network parameters (**u***-s, **w***-s, and **b***-s). Like VAE models, the model is trained straightforwardly by optimizing ELBO, as described in section III.C. The size of each input sequence data (e.g., $x_{t-T:t}$) in the training dataset is T + 1. For the *l-th* sample $z_{t-T:t}^{(l)}$, where $q \leq l \leq L$ and L is the sample length, the loss function can be formulated as [28]:

$$\tilde{\mathcal{L}}(x_{t-T:t}) \approx \frac{1}{L}\Sigma_{l=1}^{L}[log(p_\theta(x_{t-T:t}|z_{t-T:t}^{(l)}) + log(p_\theta(z_{t-T:t}^{(l)} - log(q_\theta(z_{t-T:t}^{(l)}|x_{t-T:t})))]] \tag{3}$$

For each sample, the first term of Eq.3 is the negative reconstruction error: $log(p_\theta(x_{t-T:t}|z_{t-T:t})) = \Sigma_{i=t-T}^{t} log$ $(p_\theta(x_i|z_{t-T:i}))$, and the posterior probability of x_i can be calculated as: $p_\theta(x_i|z_{t-T:i}) \sim \mathcal{N}(\mu_{x_i}, \sigma_{x_i}^2 I)$. The sum of the second and third terms is regularization (i.e., Kullback-Leibler loss). The second term $log(p_\theta(z_{t:T-i})) = \Sigma_{i=t-T}^{t} log(p_\theta(z_i|z_{i-1}))$, where z_i can be obtained by Linear Gaussian SSM initialized with the standard multivariate normal distribution. The third term is to approximate the true posterior distribution of z_i in the $z-$space in the *qnet*: $-log(q_\theta(z_{t-T:t}|x_{t-T:t})) = -\Sigma_{i=t-T}^{t} log(q_\theta(z_i|z_{z_i-1}, xt-T:t).z_i$ (i.e., z_i^k) is transformed through planar NF. $Z_i^K = f^K(F^{K-1}(\ldots f^1(z_i^0)))$, where $z_i^0 = \mu_{z_i} + \xi_i\sigma_{z_i}$, $\xi_i \sim \mathcal{N}(0, I)$, and f^k are invertible mapping functions [28].

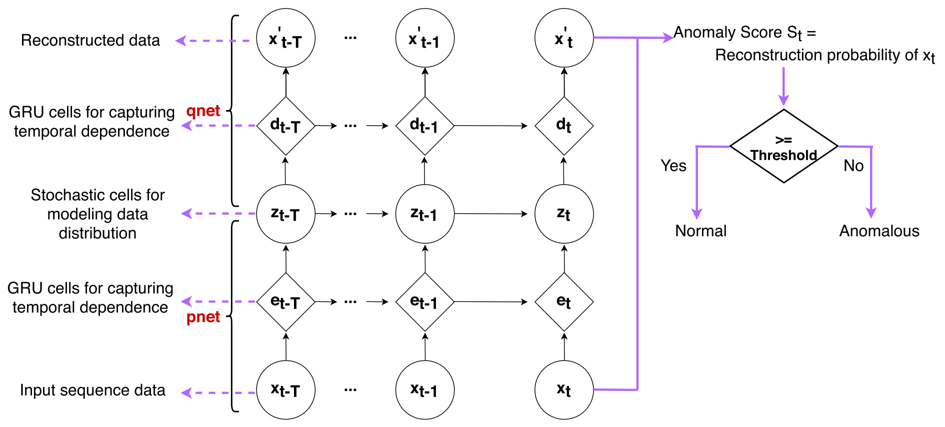
Reconstructed data
GRU cells for capturing temporal dependence
qnet
Stochastic cells for modeling data distribution
GRU cells for capturing temporal dependence
pnet
Input sequence data
x't-T
x't-1
x't
dt-T
dt-1
dt
zt-T
zt-1
zt
et-T
et-1
et
xt-T
xt-1
xt
Anomaly Score St = Reconstruction probability of xt
>= Threshold
Yes
No
Normal
Anomalous

FIGURE 6. **Overall model architecture is composed of a *qnet* and *pnet* (written in red), in which nodes refer to different variables. At time t, x_t is the input observation; x'_x is the reconstruction of x_t; e_t and d_t are memory variables in GRU cells that are deterministic; z_t is a $z-$space variable that is stochastic, and edges represent the dependence between variables.**

5) ONLINE DETECTION

We now can use the trained model to decide if an observation at a time step (say t, denoted as x_t) is abnormal or not. Note that the input of our model is a sequence data of length $T + 1$. Therefore, we take the sequence $x_{t-T:t}$, i.e., x_t and T consecutive observations preceding to it, as an input to reconstruct x_t [28]. As recommended by [35], this reconstruction, which can be evaluated by the conditional probability is used as the anomaly score in our model. The anomaly score of x_t is denoted as S_t, so $S_t = log(p_\theta(x_t|z_{t-T:t}))$ [28]. A high score indicates the input x_t is well reconstructed. If an observation follows the normal patterns of time series, it can be reconstructed with high confidence. Meanwhile, the smaller the score, the less likely it is to be abnormal [28]. Formally, if S_t is lower than an anomaly threshold then x_t is marked as anomalous; otherwise, x_t is normal, as shown in Figure 6. Next, we show how to dynamically set the anomaly threshold offline.

6) DYNAMIC THRESHOLD SELECTION

As presented in Figure 5, while the during model training module is operating, we first calculate an anomaly score for every observation with a multivariate time series of N' observations. Then all anomaly scores produce a univariate time series in the form of $S_1, S_2, \ldots, S_N$. Next, we set the anomaly threshold of th_F according to the principle of extreme value theory (EVT) [36].

EVT is a statistical theory whose aim is to find the law of extreme values, and extreme values are normally put at the tails of a probability distribution. The benefit of EVT is that when finding extreme values, it makes no assumption about the distribution of data. Peaks-over-threshold (POT) [31] is another theorem, which is used to fit the tail portion of a probability distribution by a using Pareto distribution (GPD) with parameters. POT is also adopted to learn the threshold of anomaly scores. Normally, values at the high end of a distribution are the main focus of other POT applications; however, in this study [28], anomalies are placed at the low end of the distribution. Thus, the GPD function is as follows:

$$\overline{F}_{(s)} = P(th - S > s|S < th) \sim (1 + \frac{\gamma^s}{\beta})^{-\frac{1}{\gamma}} \tag{4}$$

where γ and β are shape and scale parameters of GPD, th is the initial threshold of anomaly scores, and S is any value in $S_1, S_2, \ldots, S_{N'}$. The portion below a threshold th is denoted as $th - S$, and it is empirically set to a low quantile. Next, the values of parameters $\hat{\gamma}$ and $\hat{\beta}$ are estimated using maximum likelihood estimation (MLE) [28]. The final threshold th_F is then computed by

$$th_F \simeq th - \frac{\hat{\beta}}{\hat{\gamma}}((\frac{qN'}{N'_{th}})^{\hat{\gamma}} - 1) \tag{5}$$

where N' is the number of observations, q is the desired probability to observe $S < th$, and N'_{th} is the number of S_i s.t. $S_i < th$. For the POT method, two parameters that need to be tuned are low quantile and q. Their values are $< 7\%$ and 10^{-4} [28], respectively.

7) ANOMALY SCORE COMPARISON FUNCTION

This function compares the calculated anomaly score and calculated threshold, if $S_t >= th$ is false, then the traffic is declared as an attack source; therefore, it will be forwarded to a mitigation agent module. Conversely, if $S_t >= th$ is true, then the traffic is declared as a normal source; therefore, the algorithm continues its loop.

8) MITIGATION AGENT

If the traffic is labeled as an attack source in the previous step, then in order to mitigate the compromised switch attacks, the mitigation agent sends a *flow_mod* message attached to a *delete* action to the edge OpenFlow switch and requests the forwarding engine of the SDN controller to drop *packet_in* messages of the attacking sources (as illustrated in Figure 5).

V. EXPERIMENTAL SETUP

First, we describe attack scenarios that are deployed in this work. Second, we give readers a description of our training dataset and of training the core module of ECSD. Third, we present an elaborate implementation of the proposed scheme.

A. ATTACK SCENARIOS

To deploy compromised switch attacks, we replicate some of the attacks mentioned by the authors of both [25] and [26] as follows:

1) NETWORK-TO-Switch(S) (NtS)

One or more forwarding devices dispatch a large amount of traffic to a specific OpenFlow switch by installing custom rules for the purpose of flooding the switch. This kind of attack brings down a target switch in serious cases, and the consequence is even more disastrous in such situations such as dealing with mission-critical systems.

2) NETWORK DoS (NDoS)

Compromised switches direct traffic into a loop and enlarge a flow until it completely fills out the available link bandwidth causing a DoS. This attack relates a compromised switch that either generates, misroutes, or replays packets, which damages the whole network topology.

3) TCAM EXHAUSTION (TCAME)

First, malicious hosts may forward an unreasonable amount of traffic and compel the controller to install many flow rules. Second, TCAM is a fast-associative memory designed to store flow rules. These two reasons result in exhausting the switch's TCAM utterly. This may trigger significant latency or packet drops.

4) SWITCH BLACKHOLE (SBlackhole)

A blackhole is a network condition where the flow path ends abruptly, and the traffic cannot be forwarded as intended to the destination. In this case, a compromised switch either drops or delays packet forwarding to launch this attack, thereby preventing the flow from reaching the destination.

B. TRAINING DATASET

The server machine dataset (SMD) is one of the most credible datasets, and it is applied in many multivariate time-series studies [28], [37]. This new dataset collects diverse real network traffic types from a large Internet company, and it was publicly published in [28]. In our study, this dataset was used to train the models of ECSD. Specifically, the SMD dataset includes anomaly traffic with a ratio of 10.72%, and the training and testing set sizes are 58317 and 73729, respectively. Moreover, we collected more 30,000 data samples, both anomalous and normal traffic, recorded in our simulated network for the training set.

C. SENSITIVITY ANALYSIS OF USER-DEFINED PARAMETERS AND MODEL SETUP

In order to reduce the number of experiments and obtain the best performance for the proposed algorithm, its user-defined parameters are beneficial to investigate. There are four major key parameters in the proposed scheme, i.e., GRU layers and dense layers, $z-$space), planar NF length, POT parameters. The Taguchi's method [38] is used in our experiments to acquire a reasonable combination among factors rather than the whole combination. It applies fractional factorial test designs called OAs that serve to reduce the number of experiments. The selection of a suitable OA depends on the number of control factors and their levels. Using OA design can estimate multiple process variables which is simultaneously affecting the performance characteristic, while minimizing the number of test runs. The number of levels for four factors are set as follows: *GRU layers and dense layers* $\in \{200, 350, 500, 650, 800\}$; *(z-space)* $\in \{1, 2, 3, 4, 5\}$; *planar NF length* $\in \{10, 20, 30, 40, 50\}$; *POT parameters* $\in \{0.0001, 0.0025, 0.005, 0.0075, 0.01\}$. A traditional full factorial design would require 5^4 or 625 experiments; however, in the Taguchi's method, the required experiments are only 25 experiments because the Taguchi's method adopts an orthogonal array $L_{25}(5^4)$.

To train the model training module of ECSD, we strictly followed the descriptions in [28] and obtain acceptable user-defined parameter settings according to our experimental results. In addition, our model hyper-parameters are described as follows. The length of input data sequence was set to 100 (i.e., $\mathrm{T}+1=100$). The GRU layers and dense layers have 500 units. The ϵ in the standard deviation layer was set to 10^{-4}. Based on the sensitivity analysis above, the length of the planar NF was 20 and the dimension of $\mathbf{z}$-space variable was fixed to 3. Next, the batch size was set to 50 for training and was run for 20 epochs with early stopping. We used an Adam optimizer for stochastic gradient descent with an initial learning rate of 10^{-3} during model training. When backpropagating gradients through the network layers, gradient values may grow extremely large such that model parameters overflow (i.e., become *float.nan*) [28]. To address this issue, gradient clipping by norm was used with 10.0 as the limit. L2 regularization with a coefficient of 10^{-4} was applied to all layers of our model. During training, 15% of the training data is used for validation. For POT parameters, 0.0001, 0.0025 and 0.005 were used for the three subsets of SMD. The model parameters are given in Table 3.

D. SDN-BASED CLOUD IMPLEMENTATION AND TEST PREPARATION

Figure 7 shows the testbed for virtual networks to simulate a compromised switch attack in a production environment that consists of a router (HP MSR930–JG511A), a physical switch (HP 3500 yl), an OpenStack platform (one controller node, one network node, an SDN controller (ONOS), and three compute nodes running OvS drivers on OpenFlow virtual

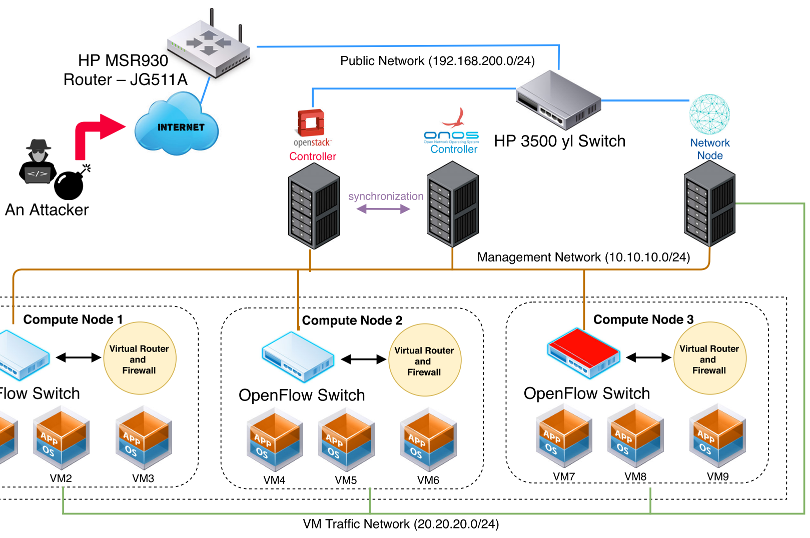

FIGURE 7. Experimental SDN-based cloud topology, including an OpenFlow switch, being compromised by an attacker.

TABLE 3. Model Parameters.

data sequence length	100 (i.e., T + 1 = 100)
GRU layers and dense layers	500 units
ϵ in the standard deviation layer	10^{-4}
z-space variables dimension	3
planar NF length	20
batch size	50
optimizer	Adam
epochs	20 (with early stopping)
learning rate	10^{-3}
gradient clipping by norm	10.0 as limit
L2 regularization	10^{-4} coefficient
validation	15% of the training data
q	10^{-4}
low quantile	0.0001, 0.0025, 0.005 for three subsets of SMD

switches for cloud networking), the Internet connection and botnets. Moreover, a synchronized connection was established between the OpenStack controllers and the SDN for the cloud information and modifications that are needed.

To simulate the compromised switch attacks, we manipulated the ONOS controller to perform the attacks mentioned in section V.A. We coded each of the attack scenarios using Python, sFlow [44] and sFlow-RT [43] were used to collect data. As we stated in section IV.A, our modules were located in the SDN application layer of the ONOS server (Figure 4). However, for reducing time in the training phases, we trained our models in a distributed manner on three different nodes: the ONOS server, network node and the OpenStack controller using an Apache Spark cluster [45]. The system configuration of each node is a dodeca-core CPU (Intel® Core™ i7-8700 CPU @ 3.20GHz), and a 48GB memory, equipped with a Gerfore 1060 GPU with 3GB onboard memory, running 64 bit Ubuntu Linux v18.04.

VI. RESULT ANALYSIS

This section contains a comprehensive analysis along with detailed comparisons between our proposed scheme and other solutions. First, we present the detection performance evaluation. Second, the attack mitigation performance is analyzed, and then the resource usage is provided. Lastly, we briefly discuss the overall performance of our proposed scheme and present the deployment discussion.

A. SOLUTIONS FOR COMPARISON

To demonstrate the performance of our ECSD scheme, we carried out experiments to compare our work with other different multivariate time-series anomaly detection algorithms and some of the existing studies by other authors on the same environmental set-up. First, we compared the two machine-learning-based methods that use a multivariate time-series technique, which are One-Class SVM [39] and long-short-term memories (LSTMs) [37]. The former algorithm is unsupervised learning that has the advantage of being able to use a very small training set to learn a classification function, which is a lightweight approach. One-Class SVM normally leads to a high false-alarm rate and poor accuracy. Next, the LSTM approach, which is also a deep-learning-based technique, gives better predictions; however, its performance is still low overall and with high resource usage. Second, we reproduced the existing work, WedgeTail [26], to compare with our proposed scheme. WedgeTail is a recent study that is designed to secure the SDN data plane. This study is a novel study and produces fairly good performance overall. Moreover, it uses various measurements that are suitable for comparison. Last, our previous work, ARIMA [16], was chosen to demonstrate the expected improvement of ECSD.

Note that we implemented these solutions, from [16], [26], [37], and [39], in our testbed according to the above observations to retain their original novelties and functionalities.

B. F1-SCORE, DETECTION RATE, ACCURACY, AND FALSE-ALARM-RATE PERFORMANCE

To properly evaluate the performance of our proposed scheme, we closely follow and adopt some criteria, such as

TABLE 4. Anomaly detection performance among 5 different solutions on average in detail.

	Detection Rate (%)	Accuracy (%)	False-Positive Rate (%)	F1-score (%)
ECSD	98.51	96.99	1.02	97.05
One-Class SVM [39]	95.65	94.14	4.34	94.20
LSTM [37]	97.06	95.69	1.98	96.18
ARIMA [16]	96.35	94.88	3.76	95.01
WedgeTail [26]	97.48	95.97	1.62	95.95

detection rate, accuracy, false alarm rate ($\overline{FAR}$), and F1-score, which were proposed in [40]. The criteria were calculated using 4 distinct measurements, true positive (TP), true negative (TN), false positive (FP), and false negative (FN). Specifically, TP is the probability of abnormal traffic that is classified as illegitimate traffic, TN is the probability of legitimate traffic that is trusted as legitimate, FP reflects the probability of normal traffic that is accepted as abnormal traffic, and FN reveals the probability of illegal traffic that is recognized as legitimate traffic. In addition, in this work, four vital criteria including detection rate, accuracy, false alarm rate, and F1-score are used to evaluate the performance of our scheme as follows:

- Detection Rate or Recall ($\overline{R}$): is the ratio of the number of detected malicious flows to the number of all malicious flows:

$$\overline{R} = \frac{TP}{TP + FN} \tag{6}$$

- Accuracy ($\overline{AC}$): is the percentage of true detection over the total the number of flows:

$$\overline{AC} = \frac{TP + TN}{TP + TN + FP + FN} \tag{7}$$

- False Alarm Rate ($\overline{FAR}$): is the proportion of abnormal flows incorrectly identified as normal flows:

$$\overline{FAR} = \frac{FP}{FP + TN} \tag{8}$$

  Therefore, this score takes both FPs and FNs into account.:

- F1-score (F1): is the weighted average of $\overline{P}$ and $\overline{R}$.

$$F1 = \frac{2 * \overline{P} * \overline{R}}{\overline{P} + \overline{R}} \tag{9}$$

  where $\overline{P} = \frac{TP}{TP+FP}$

In summary, a detection rate is the ratio of properly identified attacks over the total number of attacks that occurred in the networks. $\overline{FAR}$ indicates the ratio of wrongly identified attacks to the total amount of wrong identifications, whereas accuracy reflects how accurate the detection engine is. F1-score represents the weighted average of $\overline{P}$ and $\overline{R}$, which takes both FPs and FNs into account. All indices were measured in trials during different attack scenarios.

Figure 8, compares the five different solutions: ECSD, One-Class SVM, LSTM, ARIMA, and WedgeTail. The One-Class SVM solution performed poorly in terms of detection rate, accuracy, F1-score, and it also had the highest $\overline{FAR}$. ECSD achieved the highest scores in detection Rate, accuracy, F1-score, and it had the lowest $\overline{FAR}$. Three other solutions ARIMA, LSTM, and WedgeTail produced similar results of all four criteria, with the LSTM slightly outperforming the other two schemes. Table 4 shows the detection rate, accuracy, false-positive rate, and F1-score on average based on formulas [$6 - 9$]. Regarding the detection rate and accuracy, our novel scheme consistently accounts for the highest rates, 98.51%, and 96.99%, respectively, which clearly outperformed One-Class SVM with 95.65% and 94.14%, and ARIMA with 96.35% and 94.88%. Meanwhile, the LSTM and WedgeTail approaches produced respectively similar results of all four evaluation criteria, which vary around 97% for detection rate and around 95% for accuracy. The results for WedgeTail are high overall, but they still produce poorer performance than our proposed scheme. Regarding the $\overline{FAR}$, our proposed scheme, at 1.02%, was much better than One-Class SVM, at 4.34%, and ARIMA, at 3.76%. The LSTM and WedgeTail approaches produced fewer wrong warnings than One-Class SVM and ARIMA; however, they still had higher $\overline{FAR}$s than did ECSD. Finally, the F1-score is an excellent evaluation criterion to fully evaluate the effectiveness of ECSD because this score takes both FPs and FNs into account. As illustrated in Table 4, One-Class SVM and ARIMA the lowest F1-scores, of 94.20% and 95.01%, respectively. ECSD had the highest F1-score, 97.05%. WedgeTail and LSTM both performed well overall (around 95%), but their F1-scores were still lower than ECSD.

In Figure 8, in terms of comparison of among four distinct attack scenarios, all detection schemes performed well on the Nts, NDoS and TCAME attack scenarios, which shows that all the schemes can easily detect some kinds of flooding attacks, such as flooding a switch by adding either packets or flows. However, the schemes performed poorly on an SBlackhole attack scenario, which indicates that all the schemes have difficulty in detecting drops or packet delays.

Through the comprehensive analyses above, we can easily see the effectiveness of applying multivariate time-series techniques to detect compromised switches.

C. ATTACK MITIGATION PERFORMANCE

Assessing the practicality of every networked system is very important, so we now proceed to analyze the results recorded from our experiments among the five compared solutions.

1) AVERAGE DETECTION TIME OF A NEW ATTACK

To assess how fast a new attack can be detected, we simulated different compromised switch attacks as presented in

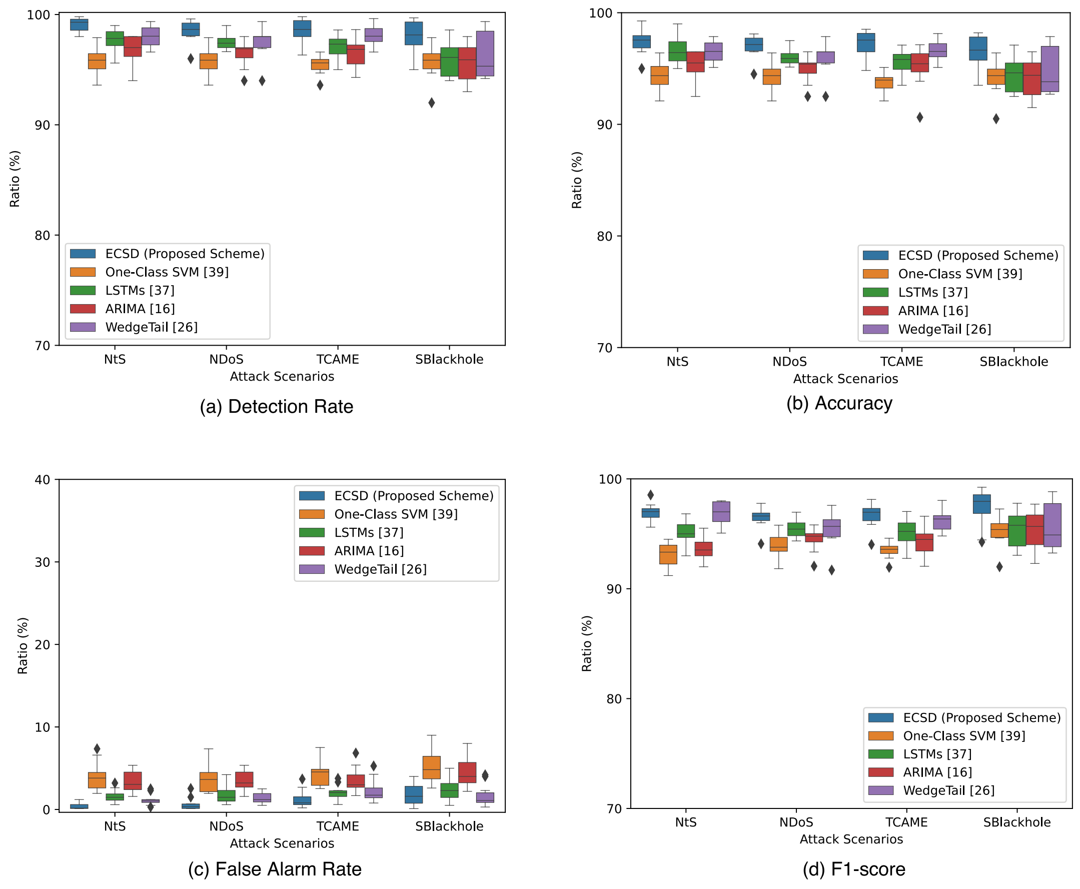

FIGURE 8. **Anomaly detection performance comparison among 4 different attack scenarios.**

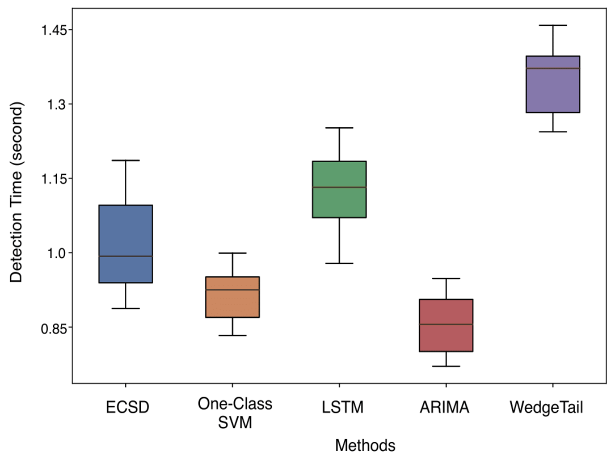

FIGURE 9. **Average new attack detection time for different solutions.**

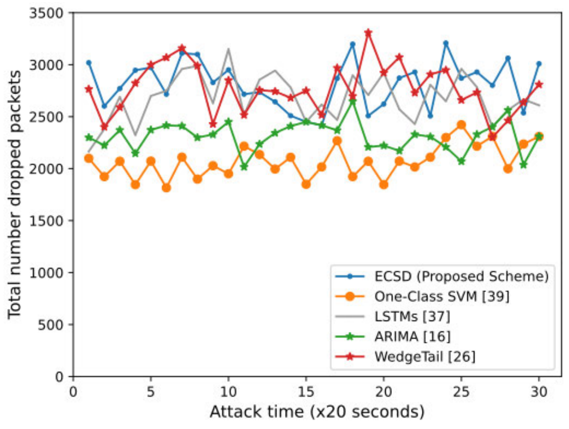

FIGURE 10. **Number of dropped packets over the duration of attack time for different approaches.**

section V.A, then created a function specialized in calculating the average detection time of a new attack, as shown in Figure 9. It is obvious that the ECSD scheme would require a longer detection time than One-Class SVM and ARIMA solutions to detect a new incoming attack in the simulated network. This is because both the latter solutions are comparatively lightweight, due to using a very small amount of historical data for training, as analyzed in section VI.A. Both ECSD and LSTM are deep-learning-based solutions, but LSTM takes around 0.15 seconds longer than ECSD to raise an alarm. This can be explained by the optimized algorithm of the ECSD and the practical use of a data pre-processing module to make it run faster. Lastly, WedgeTail takes a large amount of time to raise a detection flag due to a highly complex architecture with many policies. In summary,

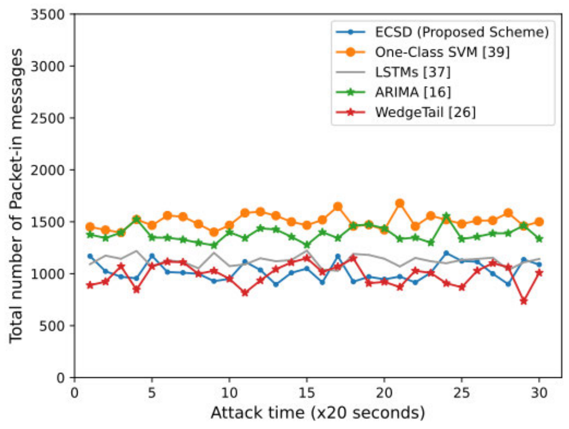

FIGURE 11. Number of packet_in messages to the SDN control plane during attacks for different approaches.

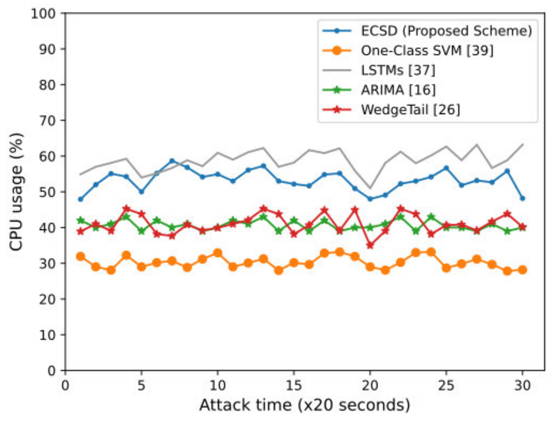

FIGURE 12. Average CPU usage.

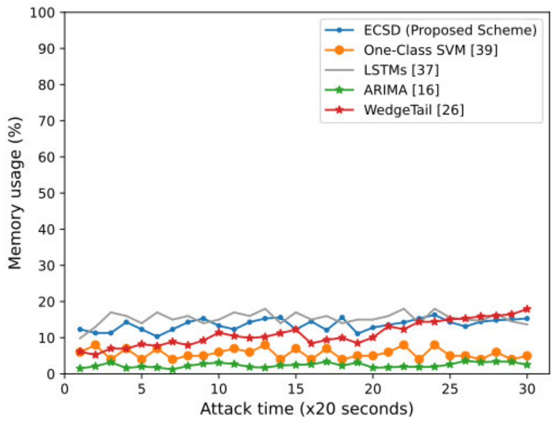

FIGURE 13. Average memory usage.

the detection time depends largely on how lightweight or heavyweight a scheme is.

2) NUMBER OF DROPPED MALICIOUS PACKETS

To protect OpenFlow switches effectively against attacks, one key evaluation criterion is how many malicious packets are dropped during the attack. Hence, the total number of dropped malicious packets when the compromised switches are under attack was collected, and the results are shown in Figure 10. Evidently, owing to the high detection rates, ECSD, LSTM, and WedgeTail conduct and implement policies as soon as an attack pattern is recognized leading to a high number of dropped attack packets. In the case of One-Class SVM and ARIMA, they are not as efficient in detecting anomalies as the three solutions above; therefore, they dropped a low number of malicious packets during the attack.

3) NUMBER OF PACKET_IN MESSAGES TO THE SDN CONTROL PLANE

Compromised switch attacks can not only produce harmful effects on the data plane but can also have damaging implications on the control plane. In our experiments, we considered the number of packet_in messages to the SDN controller as a measure of this criterion. As illustrated in Figure 11, the number of packet_in messages for different solutions show opposing results to the detection rate. Because One-Class SVM has the lowest detection rate, this triggers many flow mismatching events, which leads to the highest number of packet_in messages being created and forwarded to the ONOS controller, compared to the other four solutions. In contrast, ECSD and WedgeTail show a reasonable packet_in rate to the controller plane, and this is due to a reasonable level of malicious traffic detection and fast implementation of policies in the data plane, which protects the SDN control plane against a packet_in flooding attack [41] with high-rate attacks.

D. RESOURCE CONSUMPTION

In Figure 12, as stated in section VI, due to the lightweight character of One-Class SVM, it consumes a low amount of CPU, around 29% on average; whereas, both deep-learning-based solutions (ECSD and LSTM) use a higher amount of CPU, around 52% and 55%, respectively. Regarding memory usage, as shown in Figure 13, all schemes consume only a low amount of memory. ARIMA memory usage fluctuates between 1% and 2.6% because it uses little historical data for training. Both the deep-learning-based solutions consume a higher amount of memory; however, it is still low overall.

E. GENERAL DISCUSSION

Through the comprehensive analyses above, we can summarize some remarkable points that prove the practicality and effectiveness in reality of applying multivariate time-series techniques (LSTMs [37], One-Class SVM [39], and especially ECSD) to detect compromised switches deployed in our practical experimental setup:

- All the multivariate time-series techniques not only yield high detection rates, accuracies, and F1-scores but also produce low false-alarm rates, in which ECSD outperforms the other schemes.
- All the multivariate time-series techniques can detect a new attack in a short time.

- LSTMs and ECSD solutions drop the number of packet_in messages reasonably.
- LSTMs and ECSD solutions consume a low amount of memory.

We also notice that there is a trade-off between detection performance and CPU utilization, which explains why the detection performance of our proposed scheme ECSD outperformed the other schemes (Figure 8). ECSD requires more resources to operate, but the resource consumption is still low overall, so the trade-off is acceptable. Thus, our proposed scheme, ECSD, is an effective solution for detecting compromised switches in all metrics and in reality.

F. DEPLOYMENT DISCUSSION

To assess how sensitive are the results to the super-parameter settings for some tested algorithms, we have some comments as follows. The dimension of **z**-space plays a crucial role in ECSD, a large value would make dimension reduction have marginal effect so the reconstruction probability is impossible to discover a good posterior, and too small of it may emerge under-fitting phenomenon. In LSTM [37], values of **z**, which are an ordered set of positive values representing the number of standard deviations $\mu(e_s)$, depend on context. In our experiments, we found a range of between 4 and 10 to work well. If the values for **z** less than or equal to 3, then the LSTM model resulted in high false-alarm rates.

VII. CONCLUSION

In this paper, we proposed an effective and practical scheme to handle compromised switch attacks in an SDN-based cloud environment. This proposal not only protects OpenFlow switches on a cloud from traffic loss, traffic misrouting, traffic delays, and so on, resulting in compromised switch attacks in both the control and data planes, but also brings a better quality of service to cloud providers and customers. We presented a new approach to detect compromised switches using a multivariate time-series detection technique. In our comprehensive analysis, the approach proved highly practical and effective. We also proposed an intelligent scheme called ECSD that applied deep learning techniques to detect the anomalous behaviors that occurred in compromised switches. Moreover, the proposed dynamic threshold selection dynamically sets thresholds, which makes the scheme very practical because it can adapt to changes. For our future work, we plan to create a scheme using an algorithm known as multi-tasking, which allows one algorithm to learn multiple multivariate time-series models simultaneously. This can reduce resource consumption overall. In addition, we intend to compare our proposal with other existing works using more evaluation criteria.

REFERENCES

[1] J. Rubio-Loyola, A. Galis, A. Astorga, J. Serrat, L. Lefevre, A. Fischer, A. Paler, and H. Meer, "Scalable service deployment on software-defined networks," *IEEE Commun. Mag.*, vol. 49, no. 12, pp. 84–93, Dec. 2011.

[2] Y. Li and M. Chen, "Software-defined network function virtualization: A survey," *IEEE Access*, vol. 3, pp. 2542–2553, 2015.

[3] *Service Function Chaining*. Accessed: Aug. 2019. [Online]. Available: https://datatracker.ietf.org/wg/sfc/documents/

[4] Y. J. Zhu and L. Jacob, "On making TCP robust against spurious retransmissions," *Comput. Commun.*, vol. 28, no. 1, pp. 25–36, 2005.

[5] A. Shaghaghi, M. A. Kaafar, R. Buyya, and S. Jha, "Software-defined network (SDN) data plane security: Issues, solutions, and future directions," in *Handbook of Computer Networks and Cyber Security*. Journal of Cluster Computing, Apr. 2020. [Online]. Available: https://arxiv.org/abs/1804.00262

[6] T.-W. Chao, Y.-M. Ke, B.-H. Chen, J.-L. Chen, C. J. Hsieh, S.-C. Lee, and H.-C. Hsiao, "Securing data planes in software-defined networks," in *Proc. IEEE NetSoft Conf. Workshops (NetSoft)*, Jun. 2016, pp. 465–470.

[7] W. Yu, X. Fu, S. Graham, D. Xuan, and W. Zhao, "DSSS-based flow marking technique for invisible traceback," in *Proc. IEEE Symp. Secur. Privacy (SP)*, Berkeley, CA, USA, May 2007, pp. 18–32.

[8] S. T. Ali, V. Sivaraman, A. Radford, and S. Jha, "A survey of securing networks using software defined networking," *IEEE Trans. Rel.*, vol. 63, no. 3, pp. 1086–1097, Sep. 2015.

[9] S. Teng, N. Wu, H. Zhu, L. Teng, and W. Zhang, "SVM-DT-based adaptive and collaborative intrusion detection," *IEEE/CAA J. Autom. Sinica*, vol. 5, no. 1, pp. 108–118, Jan. 2018, doi: 10.1109/JAS.2017.7510730.

[10] Y. Duan, W. Li, X. Fu, Y. Luo, and L. Yang, "A methodology for reliability of WSN based on software defined network in adaptive industrial environment," *IEEE/CAA J. Autom. Sinica*, vol. 5, no. 1, pp. 74–82, Jan. 2018, doi: 10.1109/JAS.2017.7510751.

[11] H. Yuan, J. Bi, M. Zhou, and K. Sedraoui, "WARM: Workload-aware multi-application task scheduling for revenue maximization in SDN-based cloud data center," *IEEE Access*, vol. 6, pp. 645–657, 2018, doi: 10.1109/ACCESS.2017.2773645.

[12] M. Paliwal and D. Shrimankar, "Effective resource management in SDN enabled data center network based on traffic demand," *IEEE Access*, vol. 7, pp. 69698–69706, 2019, doi: 10.1109/ACCESS.2019.2919348.

[13] R. Ghannam and A. Chung, "Handling malicious switches in software defined networks," in *Proc. NOMS-IEEE/IFIP Netw. Oper. Manage. Symp.*, Apr. 2016, pp. 1245–1248.

[14] P.-W. Chi, C.-T. Kuo, J.-W. Guo, and C.-L. Lei, "How to detect a compromised SDN switch," in *Proc. 1st IEEE Conf. Netw. Softwarization (NetSoft)*, London, U.K., Apr. 2015, pp. 1–6.

[15] S. Gao, Z. Li, B. Xiao, and G. Wei, "Security threats in the data plane of software-defined networks," *IEEE Netw.*, vol. 32, no. 4, pp. 108–113, Jul. 2018.

[16] P. T. Dinh, T. Lee, T. N. Canh, S. P. Dang, S. C. Noh, and M. Park, "Abnormal SDN switches detection based on chaotic analysis of network traffic," in *Proc. 25th Asia–Pacific Conf. Commun. (APCC)*, Ho Chi Minh City, Vietnam, Nov. 2019, pp. 250–255.

[17] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an open, distributed SDN OS," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 1–6. [Online]. Available: https://onosproject.org/

[18] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[19] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *Proc. NIPS, Deep Learn. Represent. Learn. Workshop*, 2014. [Online]. Available: https://arxiv.org/abs/1412.3555

[20] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," 2013, *arXiv:1312.6114*. [Online]. Available: https://arxiv.org/abs/1312.6114

[21] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 153–1538.

[22] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proc. Eur. Symp. Artif. Neural Netw.*, Belgium, vol. 23, 2015.

[23] O. Ogunmolu, X. Gu, S. Jiang, and N. Gans, "Nonlinear systems identification using deep dynamic neural networks," 2016, *arXiv:1610.01439*. [Online]. Available: http://arxiv.org/abs/1610.01439

[24] A. Kamisiński and C. Fung, "Flowmon: Detecting malicious switches in software-defined networks," in *Proc. Workshop Automated Decis. Making Act. Cyber Defense (SafeConfig)*, 2015, pp. 39–45.

[25] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "SPHINX: Detecting security attacks in software-defined networks," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2015, pp. 1–15.

[26] A. Shaghaghi, M. A. Kaafar, and S. Jha, "WedgeTail: An intrusion prevention system for the data plane of software defined networks," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, Apr. 2017, pp. 849–861.

[27] C. Pang, Y. Jiang, and Q. Li, "FADE: Detecting forwarding anomaly in software-defined networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–6.

[28] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 2828–2837.

[29] C. Luo, J.-G. Lou, Q. Lin, Q. Fu, R. Ding, D. Zhang, and Z. Wang, "Correlating events with time series for incident diagnosis," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2014, pp. 1583–1592.

[30] T. J. Sejnowski and C. R. Rosenberg, "1987. Parallel networks that learn to pronounce english text," *Complex Syst.*, vol. 1, no. 1, 145–168, 1987.

[31] M. R. Leadbetter, "On a basis for 'peaks over threshold' modeling," *Statist. Probab. Lett.*, vol. 12, no. 4, pp. 357–362, Oct. 1991.

[32] *Message Layer Definition-OpenFlow Messages*. Accessed: 2018. [Online]. Available: http://flowgrammable.org/sdn/openflow/messagelayer

[33] M. Sharaev, A. Artemov, E. Kondrateva, S. Ivanov, S. Sushchinskaya, A. Bernstein, A. Cichocki, and E. Burnaev, "Learning connectivity patterns via graph kernels for fMRI-based depression diagnostics," in *Proc. IEEE Int. Conf. Data Mining Workshops (ICDMW)*, Singapore, Nov. 2018, pp. 308–314.

[34] M. Cassel and F. L. Kastensmidt, "Evaluating one-hot encoding finite state machines for SEU reliability in SRAM-based FPGAs," in *Proc. 12th IEEE Int. On-Line Test. Symp. (IOLTS)*, Lake Como, Italy, Jul. 2006, p. 6, doi: 10.1109/IOLTS.2006.32.

[35] H. Xu, Y. Feng, J. Chen, Z. Wang, H. Qiao, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, and D. Pei, "Unsupervised anomaly detection via variational auto-encoder for seasonal KPIs in Web applications," in *Proc. World Wide Web Conf. World Wide Web (WWW)*, Lyon, France, 2018, pp. 187–196.

[36] L. de Haan and A. Ferreira, *Extreme Value Theory: An Introduction* (Springer Series in Operations Research and Financial Engineering), 1st ed. New York, NY, USA: Springer, 2006.

[37] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2018, pp. 387–395.

[38] S. Gao, M. Zhou, Y. Wang, J. Cheng, H. Yachi, and J. Wang, "Dendritic neuron model with effective learning algorithms for classification, approximation, and prediction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 2, pp. 601–614, Feb. 2019, doi: 10.1109/TNNLS.2018.2846646.

[39] K.-L. Li, H.-K. Huang, S.-F. Tian, and W. Xu, "Improving one-class SVM for anomaly detection," in *Proc. Int. Conf. Mach. Learn. Cybern.*, Xi'an, China, Nov. 2003 pp. 3077–3081.

[40] N. Goix, "How to evaluate the quality of unsupervised anomaly detection algorithms?" Jul. 2016, *arXiv:1607.01152*. [Online]. Available: https://arxiv.org/abs/1607.01152

[41] T. G. Nguyen, T. V. Phan, B. T. Nguyen, C. So-In, Z. A. Baig, and S. Sanguanpong, "SeArch: A collaborative and intelligent NIDS architecture for SDN-based cloud IoT networks," *IEEE Access*, vol. 7, pp. 107678–107694, 2019.

[42] *Welcome to OpenStack Documentation*. Accessed: Mar. 2019. [Online]. Available: http://docs.openstack.org/

[43] *sFlow-RT*. (May 20, 2014). [Online]. Available: http://www.inmon.com

[44] *Making the Network Visible*. Accessed: Sep. 2019. [Online]. Available: https://sflow.org/

[45] Spark. (Sep. 1, 2019). *Apache Spark—Unified Analytics Engine for Big Data*. [Online]. Available: https://spark.apache.org/

PHUC TRINH DINH (Member, IEEE) received the B.S. degree in information technology from Telecommunications University, Vietnam, in 2018. He is currently pursuing the master's degree in information and communication with Soongsil University, Seoul, South Korea. His research interests include machine learning, deep learning, cloud computing, big data, software defined networks, network functions virtualization, and network security.

MINHO PARK (Member, IEEE) received the B.S. and M.S. degrees in electronics engineering from Korea University, in 2000 and 2002, respectively, and the Ph.D. degree from the School of Electrical Engineering and Computer Science, Seoul National University, Seoul, South Korea, in 2010. He was with Samsung Electronics Company Ltd., from 2002 to 2004. As a Postdoctoral Researcher, he was with Carnegie Mellon University (CMU), for two years from 2011. He is currently an Assistant Professor with the School of Electronic Engineering, Soongsil University, Seoul. Before his stint as a Postdoctoral Researcher with CMU, he was a Senior Engineer with the 3GPP LTE S/W Development Group, Samsung Electronics Company Ltd. His current research interests include wireless networks, vehicular communication networks, network security, and cloud computing.